Interlisp-VAX Users Manual

First Edition

Preliminary Draft

Raymond Bates
David Dyer
Andrea Ignatowski
Johannes Koomen
Steven Saunders
Donald Voreck

5 December 1982

USC Information Sciences Institute
Interlisp-VAX Project
4676 Admiralty Way
Marina del Rey, CA 90291

Interlisp-VAX is sponsored by DARPA under contract number MDA 903-81-C-0335.

Table of Contents

UNIX is a trademark of Bell Laboratories.

The following are trademarks of the Digitial Equipment Corportaion:

DIGITAL
PDP
VAX
VMS
TOPS-20

1. Introduction

Welcome to the world of Interlisp-VAX!

The Interlisp-VAX project was begun in mid-1979 to provide a newer, more powerful alternative to Interlisp-10 as a LISP environment suitable for research. The result is an efficient, portable, fully functional system compatible with other Interlisps and supporting a large virtual address space. Interlisp-VAX runs under two operating systems, VMS and UNIX[1] . Its implementation on the VAX, one of the most popular machines in research facilities and college campuses today, assures it of a long, productive future.

The implementation of Interlisp-VAX relied heavily on two sources: Interlisp-D, which runs on a personal computer developed at Xerox Palo Alto Research Center, and Multilisp, implemented at the University of British Columbia for the IBM-370. It is a more traditional implementation than some of the newer LISPs. It is written in Interlisp itself and C, the universal implementation language for the UNIX operating system, and took approximately six man-years of effort to complete. For a more detailed explanation of the implementation see *Implementation of Interlisp on the VAX* [1].

This manual is periodically revised to reflect the changes and improvements made to Interlisp-VAX, which is still in a stage of development. New manuals can be ordered from the address below.

USC/Information Sciences Institute
Interlisp-VAX Project
4676 Admiralty Way
Marina del Rey, CA 90291
USA

Mail can also be sent via the ARPANET to Interlisp@ISIB.

Interlisp-10 manuals can be obtained from Xerox Corporation. Write to the following address for order information.

Xerox Corporation
Electro-Optical Systems MS 384
300 N. Halstead St.
Pasadena CA 91107
USA
ATTENTION: Rachel Navarro

or send ARPANET mail to, Lispmanual@Parc-Maxc.

---

[1] specifically Berkeley VM/UNIX

2. Interlisp-10 Supplementary Manual

This chapter describes the major differences between Interlisp-10 and Interlisp-VAX. It is meant to be used as a supplement to the large *Interlisp Reference Manual* [5]; it cannot stand on its own as a document.

To facilitate the checking of information between the two manuals, this chapter is organized in the same manner as the Interlisp-10 guide, even down to the headings and section numbers; nearly every function and variable noted here is located in the equivalent chapter and section in the *Interlisp Reference Manual*. Of course, there are many areas in which there are no marked differences between the two Interlisps and thus nothing to make note of, so the section numbers, while in numerical order, are not necessarily consecutive (don't worry, you're not missing any sections!).

This chapter describes functions which have been revised for Interlisp-VAX; new functions for Interlisp-VAX which do not exist in Interlisp-10; machine-dependent or otherwise unneeded functions that don't appear in Interlisp-VAX; and other aspects in which the two Interlisps differ.

SECTION 3

DATA TYPES, STORAGE ALLOCATION, GARBAGE COLLECTION, AND OVERLAYS

3.1 DATA TYPES

For the most part, Interlisp-VAX utilizes the same data types for the same purposes as Interlisp-10. One exception is that FIXP, which represents a large integer in Interlisp-10, does not exist as a data type in Interlisp-VAX; all integer types are designated as SMALLP.

The following primitive data types are implemented by the VM:

SMALLP	Integer, from -2↑30 to +2↑30-1.
FLOATP	Unboxed 32-bit floating point number.
LISTP	Pair of arbitrary objects (list).
STACKP	Stack pointer.
STRINGP	String descriptor.
ARRAYP	Array descriptor.
CCODEP	Code object descriptor.
PDL	Spaghetti stack space.
PNAME	Character space for atom and string pnames.
VMSPACE	VM program and static data space, C stack.

All other data types are established as composites of these types.

3.2 USER DEFINED DATA TYPES

(DECLAREDATATYPE NAME DESCR FLG ALIGN) [Function]

is the user level and record package entry point that creates a new data type. It is also called by the record package to generate descriptors for ARRAYBLOCK and BLOCKRECORD record types.

NAME is the name for the new type. DESCR is a list of descriptors, including one or more of the following:

POINTER, XPOINTER, FULLPOINTER, FULLXPOINTER	Name of a pointer field.
FIXP	Name of a signed 32-bit integer field.
FLOATP	Name of an unboxed 32-bit floating point number.
WORD, SHORT	Name of an unsigned 16-bit integer.
SIGNEDWORD	Name of a signed 16-bit integer.
BYTE	Name of an unsigned 8-bit integer.
SIGNEDBYTE	Name of a signed 8-bit integer.
FLAG	Name of a field whose value is T or NIL.
(BITS n)	Name of an unsigned integer, 0<n<32.

(SIGNEDBITS n) Name of a signed integer, 0<n<32.

(BITS n offset) Allocates a field of n-bits. Offset, a constant integer, is added to n after a FETCH and subtracted from n before a STORE. This feature supports the (BETWEEN n1 n2) construct allowed by the record package's DATATYPE and ARRAYBLOCK types.

(SIGNEDBITS n offset) Same as above, but for signed integers.

All 8-, 16-, and 32-bit fields will be aligned 0, 1, or 2 bytes respectively. All pointer fields will be placed before all nonpointer fields. Gaps created by alignment constraints will be filled if possible.

FLG is set to ARRAY if declaring an ARRAYBLOCK, to T if declaring a BLOCKRECORD, and to NIL otherwise. ALIGN is the alignment request or NIL, which defaults to 2 (long word).

3.5 THE INTERLISP-10 SWAPPER

There are no swappable arrays in Interlisp-VAX. All of the swapper-related functions exist, but they operate as no-ops.

SECTION 5

PRIMITIVE FUNCTIONS AND PREDICATES

5.1 PRIMITIVE FUNCTIONS

Users importing Interlisp programs onto the VAX may find their CARs and CDRs behaving a little differently than they did on Interlisp-10. This is because Interlisp-VAX discourages the practice of taking the CAR and CDR of atoms, numbers, and other nonlists by generating error messages and entering a break when such actions are attempted. This is especially true in the case of numbers; whereas Interlisp-10 returns apparently random values for such function calls, Interlisp-VAX generates an error message in an uncompiled program and a hard interrupt in a compiled function (what the user types is underlined):

```
←DEFINEQ((TEST (X) (CAR X]
(TEST)
←(TEST 1)

ILLEGAL ARG
1

←COMPILE(TEST)
listing? STore and redefine
output file? No

(TEST (X))

(TEST redefined)
(TEST)
(TEST 1)

Hard Interrupt, signal 10 at 2389239 in TEST + 11
```

Interlisp-VAX provides the functions OLDCAR and OLDCDR, which more closely resemble the CAR and CDR of Interlisp-10 in that they will return a value (NIL) when passed an atom or a number. It is recommended for efficiency and reliability, however, that if there are instances in a program where the LISTP property of an object X is unknown, the user should take (CAR (LISTP X)) rather than (CAR X) or (OLDCAR X).

SECTION 8

FUNCTION DEFINITION AND EVALUATION

(EVALHOOK OBJ FUNCTION) [Function]

is a function designed to make it easy to step or trace the interpreter. It sets the variable EVALHOOK to FUNCTION and, whenever an EVAL is called, APPLYs that function to OBJ just before executing the EVAL itself. At some point the tracer called in this way will normally call (EVALHOOK OBJ 'MYSELF), which will do the actual evaluation and continue tracing lower levels of the OBJ.

Example:

```
(DEFINEQ (HOOK (OBJ)
   (EVALHOOK (PRINT OBJ) 'HOOK]
```

(HOOK anything) will evaluate "anything" and print each subform before it is evaluated.

EVALHOOK is found in the Lispusers package TRACEIN.

SECTION 10

ATOM, STRING, ARRAY, AND STORAGE MANIPULATION

10.1 PNAMES AND ATOM MANIPULATION

(U-CASE VAR FLAG) [Function]

works as for Interlisp-10, but care must be taken in its use (e.g., if used to enforce a system standard not appropriate for the UNIX operating system, such as U-CASE-ing a filename). Same for L-CASE.

10.2 STRING FUNCTIONS

(ALLOCSTRING LENGTH INITIALVALUE) [Function]

allocates a new string of size LENGTH characters. If INITIALVALUE is

-an atom	The string will contain the first letter in the atom.
-an integer	The string will contain the ASCII character that the integer represents.

10.3 ARRAY FUNCTIONS

(ARRAY SIZE TYPE INITIAL ORIGIN) [Function]

All features of Interlisp-10 arrays are duplicated in Interlisp-VAX, including mixed-type (pointer/integer) arrays. But Interlisp-VAX offers the user additional options, including the zero-origin and small integer array options included in Interlisp-D. All Interlisp-VAX arrays should be completely compatible with Interlisp-D arrays.

While the features of Interlisp-VAX are compatible with other implementations, the underlying representations of these features are different; notably, Interlisp-10 uses essentially identical representations for all arrays, hash arrays, and code arrays, but Interlisp-VAX does not. Thus the low-level hacking which works for other versions of Interlisp will not work for Interlisp-VAX.

In Interlisp-VAX arrays, SIZE is the number of elements, TYPE is the type of array and can be one of the following choices:

NIL	Array of double-pointers; both SETA and SETD can be used.
n	0 <= n <= SIZE; the first n elements of array are integers, the rest are double-pointers.

(n)	0 <= n <= SIZE; origin is 0, the first SIZE-n elements are integers, rest are pointers; used to create arrayblock records.
BYTE	Elements are unsigned bytes (8-bit integers).
SMALLPOSP	Elements are unsigned 16-bit integers.
FIXP	Elements are unboxed integers.
POINTER	Array of pointers; only SETA may be used.
DOUBLEPOINTER	Same as NIL.
HARRAY	Hash array; type is pointer, origin is 0; SIZE represents the number of items which can be put into the array before it becomes full.

INITIAL sets the initial value of each element in the array. ORIGIN specifies the origin of the array, either 0, 1, or NIL (default is 1).

Nearly all functions in Interlisp-10 which manipulate arrays and hash arrays are also present in Interlisp-VAX. One that is not implemented is ARRAYBEG; there is no way to generate pointers to the interior of arrays in Interlisp-VAX.

Swappable arrays do not exist in Interlisp-VAX.

10.4 STORAGE FUNCTIONS

DATA TYPES

There are 10 predefined data types in Interlisp-VAX (note that the type numbers are different from Interlisp-10):

type #	type	description
0	SMALLP	Small integers (<2↑30).
1	BIGP	Big integers (not implemented).
2	FIXP	Integers (same as SMALLP).
3	FLOATP	Floating point numbers.
4	LITATOM	Literal atoms.
5	LISTP	List cells.
6	STACKP	Stack pointers.
7	STRINGP	String pointers.
8	CODEP	Cexpr or subr objects.
9	ARRAYP	Arrays.

Many functions involving data types will accept either the type name or type number as an argument. The use of names is encouraged over numbers, however, as this will make the code more compatible across other Interlisp implementations.

GARBAGE COLLECTION

Unlike Interlisp-10, which invokes garbage collection frequently, Interlisp-VAX just assigns more storage space to a data type as it is needed and does not actually garbage collect until all of the memory available for immediate storage has been allocated and used. As storage space is allotted, usually in 64K byte sectors, Interlisp-VAX issues "memory expansion" messages of this form:

"expanding DATATYPE, nnn used, kkk before GC"

where nnn is the number of bytes currently in use by the DATATYPE and kkk is the number of bytes which can be allocated before garbage collection will be invoked. Once kkk reaches 0 and garbage collection begins, this message:

"....in GC...."

is printed. Several seconds later, when the garbage collection is finished, execution of the program continues.

The nature of Interlisp-VAX's garbage collector has an effect on SYSOUTs; see page 19.

GCMESS and GCGAG

As with Interlisp-10, the messages issued during memory expansion are controlled by the functions GCMESS and GCGAG.

(GCMESS NUMBER MESSAGE) [Function]

manipulates both the expansion and garbage collection messages, according to the following sequences:[2]

NUMBER:	1	2	3	4	5	6	7
MESSAGE:	expanding DATATYPE	,	nnn	used,	kkk	before GC	<CR>

and

NUMBER:	0
MESSAGE:	"..in GC.."

[2]DATATYPE is the description or name of the type being expanded and cannot be directly manipulated by the user.

0 manipulates the message printed upon entering garbage collection. There is also an option 8, which allows the user to have a message printed upon leaving garbage collection.

(GCMESS 1 NIL) silences the DATATYPE portion of the message.

MESSAGE for 0, 2, 7, and 8 must be a string; for 1, 4, and 6, it can be a string or the name of a function of one argument to be APPLYed to its adjacent message (1 to NIL, 4 to 3, 6 to 5).

(GCGAG MESSAGE) [Function]

also affects the expansion and garbage collection messages. MESSAGE can be one of the following:

T	Expansion and garbage collection messages are printed at the appropriate times.
NIL	Garbage collection or expansion messages are not printed.
A List	The CAR of MESSAGE is printed when garbage collection is begun, and the CDR is printed when garbage collection is finished.
atom or string	MESSAGE is printed when garbage collection begins and nothing is printed when it ends.

MAXFS and RECLAIM

A system that allows arbitrarily large usage of virtual memory will eventually degrade, so it is not necessarily wise to postpone garbage collection until absolutely necessary. Interlisp-VAX provides two ways to trigger garbage collection early, MAXFS and RECLAIM.

(MAXFS SIZE TYPE) [Function]

sets the maximum size of storage for a data type. SIZE is in bytes and TYPE can be either type name or number. When space allocated to a data type exceeds SIZE, a garbage collection is begun. Interlisp-10's MINFS exists on the VAX for historical reasons but has no effect.

(RECLAIM TYPE MAKESYSFLAG) [Function]

goes beyond its Interlisp-10 capabilities. If only the TYPE flag is used, it initializes a garbage collection of that TYPE just as in Interlisp-10. MAKESYSFLAG, useful when performing MAKESYS operations, triggers Interlisp-VAX's "frozen page" facility, which allows parts of the address space to be set aside and not garbage collected (resulting in a large savings of garbage collection time).

MAKESYSFLAG can be one of the following:

NIL	No change in freezing; collects the unfrozen part.
T	Forces garbage collection onto an "even" garbage collection boundary, appropriate for profiling or making a sysout.

n	Performs a garbage collection, then freeze those sectors of datatype TYPE in which more than n bytes are being used. This allows users to leave sectors with only a few bytes used available for recycling via garbage collection.
(n . k)	Thaws all frozen sectors back to k (which is a previous level of freeze), then performs a garbage collection and freezes using parameter n. (NIL . 0) thaws back to the original level and doesn't refreeze.

Freezing is incremental, meaning that if there are already frozen pages and the user requests another freeze without thawing any sectors, there will now be two sets of frozen pages.

Freezing can begin to take place only when the garbage collection flop is in its even state, so if the user is in an odd garbage collection flop upon requesting a freeze, a garbage collection to the even flop will occur before the freezing process starts. Garbage collection is then done twice more before the process is completed.

When garbage collections are called after a freeze, the frozen space is scanned in order to move any pointers there that may point out of frozen space.

MAKESYS uses the variable \MAKESYSGCTYPE as the argument to RECLAIM and is initially set to (SECTORSIZE/2 . 0).

MEMUSAGE

(MEMUSAGE TYPE MEMTYPE) [Function]

returns a dotted pair representing the amount of storage (allocated . used) by data type TYPE for memory space MEMTYPE. TYPE can be either the name or the number of that data type. MEMTYPE can be one of the following:

NIL	Represents user space.
FROZEN	Indicates system space.
T	Represents the sum of both system and user spaces.

MEMUSAGE can also take the following arguments, which are used by storage and the Lispusers package PROFILE:

IsDownalloc	Returns NIL if the garbage collection flop is even and T if it is odd.
StorageLimit	Gives the maximum storage that can be allocated in user space.
StorageLimit FROZEN	Shows the storage size allocated to system space.
StorageLimit T	Returns the sum of system and user storage space still available.

StorageRemaining	Gives the amount of storage remaining before a garbage collection becomes mandatory.
VmSize NIL	Shows the size of the VM in P0 space.
VmSize T	Shows the size of the VM in P1 space.

Because Interlisp-VAX invokes garbage collection when there are only a few byte sectors still remaining, functions called when a garbage collection is pending should not allocate large amounts of storage of any kind.

Stack Overflow

A "stack overflow trap" has been implemented to handle runaway recursion.

The function STACKOVERFLOWP, found in GCTRAPFORMS, checks actual stack space usage against the variable STACKOVERFLOWP, initially set to .5 megabytes. If stack space used exceeds STACKOVERFLOWP, the stack overflow trap is generated with the message "STACKOVERFLOWP on GCTRAPFORMS".

Users can change or remove the function STACKOVERFLOWP from GCTRAPFORMS and can change or rebind the value of the variable STACKOVERFLOWP.

SECTION 13

NUMBERS AND ARITHMETIC FUNCTIONS

13.1 INTEGER ARITHMETIC

Interlisp-VAX currently supports only "small" integers (SMALLP, from -2↑30 to 2↑30-1). Advantage is taken of the fact that all user virtual addresses on the VAX are always less than 2↑31, and hence all addresses in system space can be considered as encoded integers (an integer being 30 bits long, with the 31st bit as a sign bit and the 32nd bit as a system address mask, set to 1).

Two other integer types, FIXP (to represent full 32-bit numbers) and BIGP (to represent integers of any arbitrary size), may be implemented in the future.

13.5 REUSING BOXED NUMBERS IN INTERLISP-10 - SETN

SETN exists, but since Interlisp-VAX currently employs only SMALLP for integers, SETN performs the same function as the function SETQ.

13.6 BOX AND UNBOX IN INTERLISP - 10

LOC and VAG are not included in Interlisp-VAX.

SECTION 14

INPUT/OUTPUT FUNCTIONS

14.1 FILES

The UNIX operating system's facility for handling files posed two obstacles for Interlisp-VAX. First, the UNIX system does not save old files which have been updated. All new files supersede their old versions in the user's directory. Second, the UNIX system only recognizes distinct file names of 14 characters or less; any longer names are truncated by the UNIX system to fit this format.

The UNIX operating system file names are really pointers to files, which means that a file can have more than one name. Interlisp-VAX uses this feature to save old versions of files as new ones are created. When a file is updated, Interlisp-VAX designates it as a new file and gives it an appropriate version number. It also passes along to each updated file an unnumbered name, so that the latest version of the file always has two names assigned to it, one numbered and one unnumbered. The UNIX operating system recognizes the unnumbered names but not the numbered ones, so any manipulation of older versions of files must be done in Interlisp.

To be able to handle numbered files at the UNIX shell level, one must use the special versions of csh and ls provided with Interlisp-VAX. Setting the environment variable VERSIONS in the modified C shell causes the shell to pass the "true" names of files to programs rather than the names with characters truncated to 7 bits. Typing "setenv VERSIONS n", where n is an integer, allows the new ls to print the file names as "filename;versionnumber".

This method of handling version numbers is currently being improved; see Upcoming Attractions, page 50, for details.

Interlisp implements the version number by setting the high order bit of the last character of the file name. This convention limits the length of file names to 13 characters, since one character must be reserved for version numbers. However, if the file is a .v (binary) file, or if the FORCEEXT flag is set to T, the file name is truncated to the first 11 characters to allow room for the extension as well as the version number.

Should the UNIX operating system ever be modified to accommodate longer file names, this special procedure would be removed, allowing version numbers to be designated by ordinary characters.

The UNIX operating system also, by nature, distinguishes between upper- and lower-case characters in file names. Interlisp-VAX, however, recognizes either upper or lower case and will match the upper or lower case pattern of the previous version when creating new files.

OPENFILE

There is a slight bug concerning the function OPENFILE; see page 45.

MANIPULATING FILE NAMES

The directory structure of the UNIX operating system is represented by separating the names of successive directories leading to the intended file by slashes (e.g., /usr/Interlisp/foo). A user can change a TENEX or TOPS-20-type file name to this format by using the following function:

(TRANSLATEFILENAME FILENAME) [Function]

uses the alist FILENAMETRANSLATIONS to direct the translation of all or specific fields of FILENAME. The entries of FILENAMETRANSLATIONS can be any of the field names known to UNPACKFILENAME, plus the following:

FIRST	Each form in FIRST is evaluated before any other processing is done. FILENAME and FILENAMETRANSLATIONS are bound to the file name and to the whole translation list, respectively.
FULLNAME	A list of dotted pairs of full path names. If the CAR of any of these pairs matches FILENAME, the corresponding CDR is returned as the translation.
HOST, DEVICE, DIRECTORY, NAME, EXTENSION, VERSION	Each of these is a list of dotted pairs, which are matched with the corresponding fields as returned by UNPACKFILENAME. If any CAR matches the input, the CDR replaces it.
LAST	After all changes have been made, each form in LAST is evaluated.

Below is a sample showing the recommended format for setting FILENAMETRANSLATIONS.

```
(SETQQ FILENAMETRANSLATIONS
  ((DIRECTORY (DDYER . /lisp/ddyer/lisp)
              (VORECK . /lisp/voreck/lisp)
              (RBATES . /lisp/rbates/lisp)
              (IGNATOWSKI . /lisp/ignatowski/lisp)
              (LISPUSERS . /lisp/Interlisp/lispusers))
   (EXTENSION (COM . v))
   (NAME (MACHINEINDEPENDENT . machine)
         (FOO . newfoo))
   (LAST (SETQ FILENAME
           (PACKFILENAME (QUOTE VERSION) NIL (QUOTE BODY) FILENAME]
```

With this format, if FILENAME = "<IGNATOWSKI>FOO.COM.3", TRANSLATEFILENAME would produce "/lisp/ignatowski/lisp/newfoo.v".

FILENAME can also be a list of a file name's properties as returned by UNPACKFILENAME, in which case it would return that list with the changes according to FILENAMETRANSLATIONS.

Interlisp initially has the following advice on the SPELLFILE property list:

```
(ADVISE (QUOTE SPELLFILE)
        (QUOTE BEFORE)
        (QUOTE (PROG ((B (TRANSLATEFILENAME FILE)))
                   (COND ((EQ B FILE) NIL)
                         (T (RETFROM (QUOTE SPELLFILE) B)))))
)
```

To activiate it do:

```
READVISE(SPELLFILE)
```

This will advise SPELLFILE to try TRANSLATEFILENAME before making any other attempt the file name.

ADDRESSABLE FILES

Users may have to alter calls to the function FILEPOS in files imported from other machines, primarily because "end-of-line" representations differ among the various systems. The UNIX operating system's files use just a single character (a linefeed) to end each line, while TENEX files use a special end-of-line character and TOPS-20 files use two characters, carriage return and linefeed. These differences will also affect any files which rely on explicit file positions.

14.2 INPUT FUNCTIONS

The differences in "end-of-line" representations noted above also affect the functions READC, SETSEPR, SETBRK, GETSEPR, and GETBRK. Moreover, VAX computers use an 8-bit representation for characters rather than the 7-bit convention employed by the PDP-10s, a fact which could alter the way these functions are used.

14.3 OUTPUT FUNCTIONS

(PRINTCODE FUNCTION FROM TO) [Function]

prints the assembly code for a function. FUNCTION can be either an atom, in which case its function definition is used, or a code object as returned by GETD. FROM specifies the first address to be printed and TO the last. If FROM and TO are not supplied, the whole function is printed. If they are included, the header section of the code is printed only the first time PRINTCODE is called on that function. Here is an example (what the user types is underlined):

```
←DEFINEQ((TEST (X) (CAR X]
(TEST)
←COMPILE(TEST)
listing? STore and redefine
output file? No

(TEST (X))

←(PRINTCODE 'TEST)
TEST
Header:  Fntype=CEXPR  nArgs=1
 nFunctions=0  nBindings=0  nFreeVars=0 nGlobals=0  nConstants=-1
args:
0:       (X)
the code:
4/       ENTRY:   dv iv                    0 C0          =ENTRY
6/       pushl    r←nil                   DD 58
8/       pushl    ↑B12(r←ap)              DD AC  C
11/      movl     @(r←sp)+, r←retval      D0 9E 50
14/      jmp      @↑B4(r←bt)              17 BA  4      =Exit
NIL

←(PRINTCODE 'TEST 4 11)
4/       ENTRY:   dv iv                    0 C0          =ENTRY
6/       pushl    r←nil                   DD 58
8/       pushl    ↑B12(r←ap)              DD AC  C
NIL
```

The print commands in Interlisp-VAX use two new variables for controlling output, \STOPSCROLLMESSAGE and \#DISPLAYLINES. The variable \#DISPLAYLINES represents the number of lines the system will print to the screen before pausing (default is 22), and the variable \STOPSCROLLMESSAGE is the message printed when the pause occurs (default is a bell). Output resumes when the user types any character. A user can silence this facility either by setting \#DISPLAYLINES to NIL or by typing ahead during the display.

PRINTNUM

There is a bug concerning the function PRINTNUM; see page 45.

14.6 SYSIN AND SYSOUT

Each copy of Interlisp generated on the VAX takes up the maximum permitted amount of swapping space. Thus, a standard partition of 17 megabytes can hold just two copies of Interlisp at a time, since the copies do not share any pages. Likewise, each SYSOUT performed on the VAX takes up at least 4 megabytes of space with no pages shared, as SYSOUT files contain a copy of the entire virtual memory.

The VAX memory is divided into three sections: a "high" segment, used as stack space; a "low" segment, where the code is located prior to the first garbage collection; and a gap of gigabyte size in between the two segments. Interlisp-VAX uses the high segment as the alternate space for the copying garbage collector. However, the operating system does not support saving/restoring the high segment in executable files. So if garbage collections have been invoked an odd number of times, SYSOUT must perform garbage collection again before it can execute.

(SYSIN FILE '(ARG1, ARG2...)) [Function]

works the same as SYSIN, but it can now take a list of arguments (ARG1, ARG2..) to FILE. The arguments are the same type that a user can pass when invoking Interlisp at the exec level and are accessed through the function GETENV (see page 27).

SECTION 18

THE COMPILER AND ASSEMBLER

18.1 THE COMPILER

The optimizing compiler used be Interlisp-VAX used by is somewhat slower than Interlisp-10's compiler, but produces much better code. It is invoked in the same way as the compiler for Interlisp-10. Since the instruction set is different from that of the PDP-10, source files from the 10 must be recompiled when brought over to Interlisp-VAX.

The compiler uses two passes. The first pass takes the S-expression function definition, expands all macros, CLISP, records, and iterative statements, and generates an intermediate code. The second pass is devoted to optimizing the code, which is then passed to the assembler where it is converted to VAX-executable machine instructions.

The second pass in this release of Interlisp-VAX does only very simple optimizing. Future versions will feature more complex optimization.

For more information on the compiler, see *Conference Record of the 1980 LISP Conference* [3].

Users should remember to use SYSTEMTYPE when executing code specific to a particular system such as the VAX, TOPS-20, or TENEX. See Sec. 21.5, page 26 for details.

18.4 GLOBAL VARIABLES VS SHALLOW BINDING

Unlike Interlisp-10 which uses shallow binding of variable, Interlisp-VAX is a deep-bound system, which means that Interlisp-VAX there is a difference between GLOBALVARS declarations and SPECVARS declarations. Because of the nature of a deep-bound system, it takes less time to look up GLOBALVARS than to find SPECVARS. And where RESETVARS and PROG are virtually identical commands on the 10, they are different on the VAX; RESETVARS works only for GLOBALVARS, and PROG handles only SPECVARS and LOCALVARS. Failure to use these declarations correctly can lead to inefficiencies in programming.

18.7 COMPILER MACROS

To allow differential compilation of code to run on both the PDP-10 and the VAX, the Interlisp-VAX byte compiler provides several new macro properties and formats in addition to the standard Interlisp-10 compiler macros.

The byte compiler checks the set of macro properties in the order they appear on the list COMPILERMACROPROPS and uses the first non-NIL value found. There are three principal

macro properties on this list: MACRO, BYTEMACRO, and VAXMACRO. BYTEMACRO and VAXMACRO take priority over MACRO properties and are equivalent, differing only in that BYTEMACROs are used by the compiler itself and VAXMACROs are for user code. The properties work as such:

Use MACRO for macros that will affect both the PDP-10 and the VAX.

Use 10MACRO for macros that are to be used only on the PDP-10.

Use VAXMACRO for macros that should be used only by the VAX.

Besides the regular substitution, computed, and LAMBDA-type macros available on the 10, the Interlisp-VAX byte compiler offers three additional options:

T [Macro]

as a macro means that the call to this function should be compiled closed; this is a simple way of turning off other macros. For example, if a function has a MACRO property which provides necessary information for the PDP-10 but which should be ignored by the VAX, then giving the function a BYTEMACRO of T would instruct the byte compiler to ignore the MACRO property.

(= . OTHERFUNCTION) [Macro]

tells the compiler to compile this function exactly as it would compile the function OTHERFUNCTION. On the VAX, for example, FRPLACAs are treated like RPLACAs by setting a BYTEMACRO in FRPLACA's property list to (= .RPLACA).

(OPENLAMBDA ARGS BODY) [Macro]

is a cross between substitution-type and LAMBDA-type macros. The compiler attempts to substitute ARGS for the formal arguments wherever this preserves the frequency and order of evaluation that would have resulted from a LAMBDA expression. It also produces a LAMBDA binding only for those instances that require one. For example, a VAXMACRO for ILESSP might be

(OPENLAMBDA (X Y) (IGREATERP Y X))

This would force a binding in a situation such as (ILESSP (ABC) (DEF)), because (ABC) must be executed before (DEF). On the other hand, there would be no LAMBDA binding for (ILESSP (ABC) 10), since the order of execution does not matter.

PUNT

causes a compilation error.

CONSTANT

The following are "magic" numbers to TENEX, TOPS-20, and the PDP-10, and they may not have the same effect if included in Interlisp-VAX programs:

all octal constants
7 (bits per character)
36 (bits per word)
5 (characters per word)
127 (character mask)
128 (number of characters in character set)
512 (words per page)
262143 (or 777777Q) (halfword mask)
262144 (or 1000000Q)

Magic numbers for the VAX include the following:

8 (bits per character)
16 (bits per word)
32 (bits per longword)
65536 (bytes per sector--allocation unit for memory management)

18.9 BLOCK COMPILING

Block compilation as performed in Interlisp-10 does not exist in Interlisp-VAX. When BCOMPL is invoked, Interlisp-VAX compiles and keeps all functions distinct from one another rather than compiling them into one indiscernible block. In the process, it "hides" the individual functions by attaching a prefix of the form \BLOCKNAME/ to the name of each function in the "block." The "block" is then entered and controlled in the same manner as any regular block on Interlisp-10.

BCOMPL is still slightly more efficient than TCOMPL since it treats all undeclared variables as LOCALVARS.

Setting the variable MERGEBINDFLG at compile time allows the user to see LOCALVARS in a compiled function while in BREAK mode without creating bindings for them.

18.10 LINKED FUNCTION CALLS

Linked function calls are not implemented in Interlisp-VAX, so be careful not to BREAK many LISP system functions. A BREAK(READ), for example, could result in an infinite recursion. However, REVERT can still be used to back up to a function even if breaking that function would normally be unsafe.

18.12 THE BLOCK COMPILER

Because of the special way Interlisp-VAX block compiles, all BCOMPLed code, which normally runs faster than TCOMPLed code in Interlisp-10, now executes at about the same speed on the VAX as TCOMPLed code.

18.14 ASSEMBLE

The ASSEMBLE directive of Interlisp-10 is not currently implemented in Interlisp-VAX.

SECTION 20

MASTERSCOPE AND HELPSYS

20.2 HELPSYS

There is no HELPSYS in Interlisp-VAX. The current implementation of HELPSYS contains machine-dependent code which has not been updated and requires a database which is extremely out of date even for Interlisp-10.

SECTION 21

MISCELLANEOUS

21.1 MEASURING FUNCTIONS

(IDATE TIMEANDDATE) [Function]

performs similarly to Interlisp-10's IDATE. However, because the UNIX operating system uses a different time standard than the PDP-10, the number that Interlisp-VAX's IDATE returns is not the same number that would be calculated from the same date on the 10, so some Interlisp-10 programs which use arithmetic on these integers may have to be altered. Also, TIMEANDDATE must be either NIL (in which case the current time and date are passed) or the complete time and date in the form "dd-MMM-yy hh:mm:ss".

GDATE and DATE in Interlisp-VAX treat Interlisp-10's FORMATBITS argument as a no-op; its inclusion (or lack thereof) does not affect these functions' output.

Although DISMISS does exist in Interlisp-VAX, it is not as precise as DISMISS in Interlisp-10 because the UNIX operating system's clock times in intervals of seconds rather than milliseconds. DISMISS still takes milliseconds as its argument, but the time it actually pauses is rounded off to the nearest second (to one second for any argument less than a second). Typing any character stops the pause immediately.

The function GCTRAP does not exist.

(PAGEFAULTS WHICHKIND) [Function]

uses the UNIX system call vtimes to compute the page fault rate. WHICHKIND can be any of the followingn:

MINOR	Page faults incurred in simulation of reference bits.
NIL or MAJOR	Real page fault rate.
SWAP	The number of swaps which occured.

Under VMS all pagefaults are reported as MINOR.

The UNIX command control-Z temporarily suspends Interlisp and places the user at the shell level. The user can return to Interlisp by the UNIX shell command fg (foreground job). The job will continue as if there had been no interruption (except that any typeahead will be flushed).

LOGOUT is the same as control-C in Interlisp-VAX; once either command been executed there is no way to continue where the user left off. If this strategy for LOGOUT is found to be undesirable, the user can redefine it via the SETINTERRUPT function (see page 33).

21.3 INTERFORK COMMUNICATION IN INTERLISP-10

Due to their dependence on PDP-10 hardware, the functions GETBLK and RELBLK do not exist in Interlisp-VAX.

21.5 MISCELLANEOUS OPERATING SYSTEM FUNCTIONS

Again, because of the differences in the hardware and operating systems between the PDP-10 and the VAX, many of the machine-dependent functions used by Interlisp-10 do not exist in Interlisp-VAX. These include LOADAV, ERSTR, JSYS, HOSTNUMBER, and TENEX. FILDIRand USERNUMBER are also not currently part of Interlisp-VAX, but plans are under way to implement them in the future.

(SYSTEMTYPE OPTION) [Function]

returns the type of system the program is currently running on. SYSTEMTYPE on Interlisp-VAX returns "VAX". It is most useful in a SELECTQ situation such as this:

```
(SELECTQ (SYSTEMTYPE)
         (VAX (...))
         (TOPS-20 (...))
         (D (...))
         (...))
```

In addition, if OPTION is OSTYPE, it will return the type of operating system which exists on that machine; in the case of Interlisp-VAX it would return UNIX, EUNICE, or VMS.

Four additional functions (CHDIR, DIRECTORYNAME, 0SYSCALL, and GETENV) interact with the UNIX operating system:

(CHDIR TODIR ATOMORSTRING) [Function]

is roughly equivalent to cd in the UNIX shell. It is the same as CNDIR in Interlisp-10. ATOMORSTRING set to T prints the information as an atom; set to NIL, it returns the information as a string.

(DIRECTORYNAME DIR ATOMORSTRING) [Function]

returns information about the user's directory. If DIR is T, it gives the user's current working directory (same as pwd in the UNIX shell); if NIL, it returns the user's login directory. ATOMORSTRING set to T prints the information as an atom; set to NIL, it returns the information as a string.

(0SYSCALL NUMBER ARG1 ARG2 ....) [Function]

performs the UNIX system call represented by NUMBER according to the ARGX arguments.

(GETENV WHICHARG WHICHLIST) [Function]

is used to return information, either from the user's UNIX environment file or from arguments the user may have typed in upon invoking LISP. WHICHLIST can be either T (meaning return one of the arguments typed to run Interlisp) or NIL (meaning return information from the environment file). WHICHARG can be one of the following:

NIL	Return the number of arguments run with Interlisp or the number of entries in the environment file.
a number	Print the nth argument passed or the nth entry in the environment file.
a string	Return the matching environment variable (only for WHICHLIST = NIL).

Examples:

(GETENV 'USER) prints a string corresponding to the shell environment variable $USER.

(GETENV 1 T) gets the first argument to Interlisp. If LISP was invoked by "lisp foo bar" it returns "foo".

Interlisp-VAX uses two enviornment variables (see page 29):

SysLispInitFile	The file where site's initialize is.
LispInitFile	The user's initialize file is.

DIRECTORIES

In accordance with UNIX convention, the variable DIRECTORIES will recognize ../ and ./ in directory path names.

A convenient way to load subfiles in a system-independent way is to use the following construction in filecoms:

```
(FILES
   (SYSLOAD COMPILED FROM VALUEOF
   (FILENAMEFIELD (INPUT) 'DIRECTORY))
   FOO BAR )
```

This construction causes compiled versions of FOO and BAR to be loaded from the same directory that the current input file is being read from.

21.6 JFN FUNCTIONS IN INTERLISP-10

Since JFNs are TENEX phenomena, none of the functions that refer to them (GTJFN, JFNS, OPENF, OPNJFN, RLJFN) appear in Interlisp-VAX.

21.8 TYPESCRIPT FILES

There is a slight bug concerning the function DRIBBLE; see page 45.

21.11 SETALINK AND SETCLINK

Neither SETALINK nor SETCLINK is implemented in Interlisp-VAX.

SECTION 22

THE PROGRAMMER'S ASSISTANT

22.12 GREETING AND USER PROFILES

GREET searches for INIT.LISP files in the following ways:

For the system greeting, GREET looks in the file specified by the environment variable SysLispInitFile. If the variable does not exist, it looks in /lisp/Interlisp.

GREET searches for the personal greeting first in the file mentioned in the LispInitFile variable, then in either $HOME/INIT.LISP or $HOME/INIT.LSP.

The user's name, as set by his greeting, must match $USER exactly, even in case, to have first name and initials set correctly.

SECTION 24

LISPUSERS PACKAGES

Most of the machine-independent packages that exist in Interlisp-10 also appear in Interlisp-VAX. Features such as CJSYS and EDITA, which rely heavily on the TENEX or TOPS-20 operating systems, have not been included. Some packages that do not appear in the current version of Interlisp-VAX will be present in future editions. These are listed in Upcoming Attractions, page 50.

Listed below are some new Lispusers packages designed for Interlisp-VAX.

BSCAN

BSCAN allows users to monitor the look-up of free variables in any LISP sysout. It uses an alternate set of free variable look-up functions which can record each free variable reference on the FRAMESCAN property of the atom referenced. These functions are listed below.

(BSCANINIT) [Function]

sets up the monitor facility and initializes the counts.

(BSCANON) [Function]

starts recording the information.

(BSCANOFF) [Function]

stops recording the information.

(BSCANCOLLECT) [Function]

stops monitoring and collects the accumulated data.

(BSCANPRINT OPTION OPTION ...) [Function]

does a BSCANCOLLECT if necessary and prints a summary of the information. OPTION can be any of the following:

n	Limits the printing to the first n items.
GLOBAL	Prints only searches of global variables.
BOUND	Shows collected information for bound variables.
AVERAGE	Sorts by the average number of frames scanned.
CALLS	Sorts by the number of calls.

The default is to print all information, sorted by the number of frames scanned.

PROFILE

PROFILE monitors the performance of functions within a LISP environment using the UNIX PROFIL facility. It keeps track of the PC and makes note of the most heavily executed code.

(StartProfiling SCALE FNS OTHERS WHICHSPACE) [Function]

initializes the data base and starts the profiling.

Functions are monitored via a PC map divided into equal sections. A section counter is incremented each time the PC enters a particular section. SCALE is the power of two to scale the PC map by (e.g., 4 = 16 bytes per section).

FNS can be either a list of functions to monitor explicitly or T to monitor everything. OTHERS is an integer representing how many other "most used" functions the PROFIL facility keeps track of. WHICHSPACE selects which program space to watch:

VM	Only the kernel C code.
FROZEN	All of the kernel plus frozen space.
NIL	Everything.

(PauseProfiling) [Function]

suspends profiling.

(ResumeProfiling) [Function]

resumes suspended profiling. Because PauseProfiling and ResumeProfiling nest, multiple pauses must be matched by the same number of resumes.

(StopProfiling ABORTFLG) [Function]

stops the profiling facility. If ABORTFLG = anything but the word ABORT, UpdateProfileInfo is done automatically.

(UpdateProfileInfo) [Function]

empties the profile array into the permanent data base. This must be done before results can be printed.

(PrintProfileInfo NAMEDONES? HOWMANY) [Function]

prints the accumulated data. NAMEDONES? can be any of the following:

NIL	Prints the most used functions.
T	Prints the data for the specifically monitored functions.
function name	Prints the data for that function.

HOWMANY is the number of lines to print to the screen.

PrintProfileInfo produces three columns: the name of the function, the number of times it was called, and the percentage of the total time spent in that function. A (VM) after a function name indicates a C function.

Although profiling is affected by garbage collection, the fields in BEFORERECLAIMFORMS assure valid profile data even if a garbage collection occurs during profiling.

To effectively profile code in the Interlisp kernel, symbols have to be found. The original Interlisp makesys contains symbols, and the package will give you the opportunity to use them. When a sysout or makesys is made it is also possible to include symbols by linking the file to copy symbols from into the current directory with name VM (all caps) when making a new sysout.

APPENDIX 1

CONTROL CHARACTERS

MISCELLANEOUS

control-C

is the same as LOGOUT in Interlisp-VAX; it aborts Interlisp so that it cannot be reentered and continued.

control-T

appends information available from the VTIMES system call onto the regular program status information. The new printout ends with

nn% ws = kk zz pf/s

where

nn is the user's percent of the CPU;
kk is the user's average working set size, in pages;
zz is the user's number of page faults per second of CPU time.

This information is supplied to control-T by the following function:

(CONTROLTVTIMES VERBOSE) [Function]

is found in the variable CONTROL-TFORMS. Setting VERBOSE to T within CONTROL-TFORMS causes control-T to also print out the user time, system time, and compute time.

control-Z

is the UNIX control-Z that temporarily stops a job. It is the only way to exit Interlisp-VAX and be able to reenter it where one left off, using the fg command (equivalent to using control-C, then CONT on Interlisp-10).

Users can change the behavior of all interrupt characters with the functions GETINTERRUPT and SETINTERRUPT.

(GETINTERRUPT ELEMENT) [Function]

returns the value or class of valid interrupt characters (see *The Interlisp Virtual Machine Specification* [4]). ELEMENT can be any of the following:

T	Returns a list of the codes of all valid interrupt characters.
NIL	Returns the codes of all nonbasic interrupt characters.
code number	Prints the name of the interrupt class it refers to.
class name	Prints its character code number.

Below is a list of all the valid interrupt characters in Interlisp-VAX and their code numbers.

code #	class
0	RUBOUT
2	BREAK
3	\QUIT
4	RESET
5	ERROR
8	HELP
14	CTRLUFLG
15	OUTPUTBUFFER
16	PRINTLEVEL
17	\QUOTE
19	\STOPOUTPUT
20	CONTROL-T
26	\PAUSE

(SETINTERRUPT CHAR CLASS) [Function]

allows the user to change the functions of interrupt characters. CHAR is the ASCII representation of the control characters (1-26), and CLASS is one of the classes shown above. Only control characters can become interrupt characters.

3. UNIX Operating System Dependencies

3.1. Csh and Ls

In order for users to be able to read the version numbers that LISP generates when creating files, Interlisp-VAX includes slightly altered versions of csh and ls shell commands. See page 15 for details.

3.2. Increasing the Maximum Segment Size in the UNIX Operating System

To make Interlisp-VAX more useful, it may be necessary to increase the maximum size permitted for each process in Berkeley UNIX. The procedure described below will raise the limits of both the data segment size and the stack segment size of each process from ~6Mbytes to ~11Mbytes. These changes do not relax current limits on text segment size, however.

The implementation of the changes is as follows:

Definitions of constants DMMIN and NDMAP in file /sys/h/dmap.h are changed.

```
#define NDMAP 24  /* Size of the swap area map (was 16). */
#define DMMIN 128 /* The initial block size in clicks (was 32). */
```

DMMIN is increased by 400 percent to enlarge the minimum allocation (for data and stack) per process. NDMAP is increased by 50 percent to enlarge the total number of allocations per process. Thus, for data or stack segments, a process is initially given a chunk of 128 (DMMIN) blocks. If necessary, a subsequent chunk of 256, then 512, and finally 1024 blocks is given. The last chunk size of 1024 (DMMAX) block is repeatedly allocated 21 times, if necessary, until 24 (NDMAP) allocations are made in all.

Definitions of constants MAXDSIZ and MAXSSIZ in the file /sys/h/vmparam.h are changed.

```
#define MAXDSIZ (22*1024-128-SLOP) /* max data size (clicks) */
#define MAXSSIZ (22*1024-128-SLOP) /* max stack size (clicks) */
```

MAXDSIZ and MAXSSIZ are increased by 182 percent to enlarge the total allocation (for data and stack) per process. This patch must be made in conjunction with the above change to dmap.h.

MAXDSIZ and MAXSSIZ can each comprise 21 1024-block chunks, plus chunks of 128, 256,

and 512 blocks. The aggregate of the latter chunks is equivalent to 1024-128. Hence the maximums: 21*1024 + (1024-128) = 22*1024-128.

The program max.c[3] can calculate appropriate values for MAXDSIZ and MAXSSIZ as a function of DMMIN, DMMAX, NDMAP. Executing the program via

```
max dmmin dmmax ndmap
```

will provide definitions for the relevant constants. For example, using original values,

```
max 32 1024 16
```

yields

```
#define DMMIN 32
#define DMMAX 1024
#define NDMAP 16
#define MAXDSIZE (12256)-SLOP
#define MAXSSIZE (12256)-SLOP
```

Using the cited modifications,

```
max 128 1024 24
```

yields

```
#define DMMIN 128
#define DMMAX 1024
#define NDMAP 24
#define MAXDSIZE (22400)-SLOP
#define MAXSSIZE (22400)-SLOP
```

The following is the text of **max.c.**

```
main(argc, argv)
  int argc;
  char **argv;
{
  int dmmin, dmmax, ndmap;
  int thissize, totalsize = 0;
  int part;

  argc--, argv++;
  if (argc != 3)
        printf("a.out DMMIN DMMAX NDMAP\n"), exit(1);
  dmmin = atoi(argv[0]);
```

[3]Compliments of Bill Joy of the University of California at Berkeley.

```
        dmmax = atoi(argv[1]);
        ndmap = atoi(argv[2]);
        thissize = dmmin;
        for (part = 0; part < ndmap; part++) {
                totalsize += thissize;
                if (thissize < dmmax)
                        thissize *= 2;
        }
        printf("#define DMMIN %d\n", dmmin);
        printf("#define DMMAX %d\n", dmmax);
        printf("#define NDMAP %d\n", ndmap);
        printf("#define MAXDSIZE (%d)-SLOP\n", totalsize);
        printf("#define MAXSSIZE (%d)-SLOP\n", totalsize);
}
```

4. VMS Operating System Dependencies

4.1. EUNICE

This chapter describes the idiosyhoacies of running Interlisp-VAX on a VMS system. To enable Interlisp-VAX ro run under VMS, we are using a compatability package (called EUNICE) to translate UNIX system calls into VMS system calls. EUNICE runs under VMS and allows the user to run standard UNIX software on a VMS machine. Using the EUNICE package, Interlisp-VAX runs on a VMS system while "thinking" it is operating on the UNIX system. For more information about EUNICE, send mail to Untulis@SRI-AI or write the address below.

Charles Untulis
SRI International
333 Ravenswood Ave
Menlo Park CA 94025
(415)-326-6200

4.2. UNIX Environment Variables Under VMS

Interlisp needs two environment variables for the GREET function (see page 29). VMS users can use the DEFINE command to create them. The following example shows the creation of the two important environment variables:

```
$ define "SysLispInitFile" "/usr/ddyer/lisp/init/sysinit.lsp"
$ define "LispInitFile" "/usr/ddyer/lisp/init/init.lsp"
```

4.3. VMS Resources

A VMS system has many quotas and parameters. Although Interlisp-VAX can use the default values of most quotas, the values of PGFLQUOTA and ASTLM must be changed in order for Interlisp-VAX to work.

Interlisp-VAX uses the sytem resouces's PGFLQUOTA to to determine how much memory to use. If PGFLQUOTA is too big (or if it is bigger than the file SYS$SYSTEM:PAGEFILE.SYS), Interlisp might hang the entire VMS system waiting of a free page that will never occur. This also might happen if too many Interlisps are running on one system. A good PGFLQUOTA number is 30000 per Interlisp users. If the PGLFQUOTA is too small Interlisp won't run, it will die trying to do a garbage collection. If the PGFLQUOTA number is not changed by the system manager for each user of Interlisp-VAX, Interlisp might not run.

Interlisp needs the AST queue limit (ASTLM) to be about 30. If the number is too small Interlisp may generated an I/O error when trying to read from a file.

4.4. VMS Files

The VMS version of Interlisp writes a FIXED-length, 512-byte file. This is not a standard VMS text file (which has variable length records with an implied <cr><lf> after each record). To convert from one file form to the other use the program UNIXTOVMS or VMSTOUNIX. Interlisp cannot do random access on VMS text files.

(GETFILEINFO FILE ATTRIB SCRATCH) [Function]

tells what kind of VMS file you have if ATTRIB is FILETYPE. It will return the following:

"UNIX – FILE"	Standard UNIX file.
"TEXT – FILE"	VMS text file.
"VAR – FILE"	Nontext VMS variable-length record file.
"TTY – FILE"	Teletype.
"MBX – FILE"	PIPE/MAILBOX.
"DIRECTORY – FILE"	Directory.
"USER – HANDLED"	Special fileto be handled by the user.
"RAW – QIO"	Special file to be handled by VMS $QIOs.

(OPENFILE FILE ACCESS RECOG BYTESIZE MACHINE.DEPENDENT.PARAMETERS) [Function]

writes a VMS text file instead of a UNIX-style file if the MACHINE.DEPENDENT.PARAMETER of OPENFILE is (VMSTEXT).

4.5. Configuring the Emulation of the UNIX File System

Interlisp-VAX filenames are specified in UNIX form (e.g. SYS:[FOO.BAR]FUM.COM must be given as /SYS/FOO/BAR/FUM.COM). In order to do this, the program MAKEROOT must be run by someone like you system manager. The following is an excerpt from *Installng EUNICE*. [2]

The information is supposed to guide VMS sites running Interlisp-VAX that don't run EUNICE. It shows how to successfully run the program MAKEROOT and to create the file SYS$SYSTEM:ROOT. If this file isn't created Interlisp-VAX will still be able to run but it will not be able to do file I/O.

The first step is to edit the file EUNICE.COM to describe the system in which Interlisp will run. EUNICE.COM should be run each time the VMS system is started (it should be called from SYS$SYSTEM:[SYSMGR]STARTUP.COM). The first part of the file sets up the system-wide logical names required to correctly convert local time to GMT (which is the time base used by UNIX). The logical name GMT←DSP must translate to a string that specifies the displacement west of GMT. A

displacement east of GMT can be had by starting the string with a "-". The file shipped with the EUNICE distribution tape has GMT←DSP set for 8 hours west of GMT (California). Modify this line (if necessary) to reflect the displacement west of GMT for your site:[4]

```
$ define/system   GMT←DSP        "0 08:00:00.0"  !West of GMT
```

The next line in the file determines whether or not Daylight Savings Time is in effect. It should be commented out (with a "!") when DST is not in effect. Thus, the logical name DST←FLAG should not exist when Daylight Savings Time is not in effect and should translate to the string "ON" when Daylight Savings Time is in effect:

```
$ define/system   DST←FLAG       ON       !DST is in effect
```

The next set of logical names are used to mount directories at the root of the UNIX file system. For every logical name of the form NAME -> DIRSPEC the directory specified by DIRSPEC is mounted at the root of the UNIX file hierarchy as /name. There are two possible specifications for DIRSPEC. If it just consists of a device name then the root directory of the device ([0,0]) is mounted. If it consists of both a device and directory name then the specified directory (and, of course, its subdirectories) is mounted. Given the following assignments:

```
Logical Name          Translation

SYS                   DBA0:
USR                   DBA1:
```

The following UNIX to VMS name translations will occur:

```
UNIX File             Equivalent  VMS  File

/sys                  DBA0:[0,0]000000.DIR (the  [0,0]  directory  file)[5]
/sys/a                DBA0:[0,0]A.DIR  (the  [A] directory file)
/sys/a/b.c            DBA0:[A]B.C
/sys/a/b/c.d          DBA0:[A.B]C.D
/usr                  DBA1:[0,0]
/usr/a/b.c            DBA1:[A]B.C
```

In order to make the mount assignments disk drive independent (i.e. you can mount the pack on any drive on the system and things will still work) we use the logical name translations for the various packs to find out which drive they are on. By default, when a disk is mounted a logical name of the form DISK$packname is created by VMS. This name translates into the name of the device on which the pack is mounted. The logical name SYS$SYSDISK always translates to the device on which the system pack is mounted. The commands in the EUNICE.COM file are an example of how to setup the mounted file systems for a configuration that consists of 2 disk drives. One drive holds the system

---

[4] Eastern time is 5hrs west of GMT, Central time is 6hrs west of GMT, Mountain time is 7hrs west of GMT and Pacific time is 8hrs west of GMT.

[5] thus the world has to read access to [0,0]000000.DIR files on all disks.

disk, one drive contains the user files, then it is desired to mount the system pack as /sys and the user pack as /usr

```
$ assign/system    'f$log("SYS$SYSDISK") sys      !/sys
$ assign/system    'f$log("DISK$USER")'  usr      !/usr
```

You may also include this line (which just makes /lisp be the same as /usr):

```
$ assign/system    'f$log("USR")'        lisp     !/lisp (same as usr)
```

Now execute the commands in EUNICE.COM with the "@" command. Now edit the ROOT.TXT file. It should have in it a list of all the directories you want to have mounted at the root of the UNIX file hierarchy (at least one entry for each disk you want Interlisp to be able to use). The sample file supplied contains:

```
/sys
/usr
```

Run the program MAKEROOT and specify ROOT as the directory file and ROOT.TXT as the input file. MAKEROOT should not declare any errors this time (i.e. it should run silently). If it does there are two possible reasons. You may have incorrectly setup a logical name in the EUNICE.COM file causing MAKEROOT to not be able to access the specified directory file or you may have specified a directory on a device which is not currently mounted. It the problem is the former, go back to the EUNICE.COM file and fix it. If the problem is the latter, it is a good idea to mount something on the device so that MAKEROOT can generate an i-node for the specified directory (this is not absolutely necessary though). Once you are happy with the results (usually meaning that there are no messages from MAKEROOT) you must rename or copy the generated file ROOT to SYS$SYSTEM:

```
$ r makeroot
Name of directory file: root
Input file: root.txt
$ copy  ROOT. SYS$SYSTEM:ROOT
```

If you ever change your configuration you should be sure to update the EUNICE.COM file and then re-generate the root and device directories as described above (after editing the ROOT.TXT files). For a three pack file system EUNICE.COM would look like:

```
$ assign/system    'f$log("SYS$SYSDISK") sys      !/sys
$ assign/system    'f$log("DISK$USER")'  usr      !/usr
$ assign/system    'f$log("DISK$USER")'  usr2     !/usr
```

and ROOT.TXT would look like:

```
/sys
/usr
/usr2
```

In summary for every disk drive on your system have the proper assign statement in EUNICE.COM and a entry in ROOT.TXT.

The following are some recommended quotas for EUNICE users[6]. :

---

[6]We don't know if they are all actually needed for Interlisp-VAX yet.

```
PRIO  4
PRCLM 6 to 10
TQELM 30
BYTLM 10000 to 30000
FILLM 40 to 60.
```

5. Known bugs and deficiencies

The following of bugs are known to exist in this release of Interlisp-VAX and will be corrected in future editions.

5.1. DRIBBLE

The function DRIBBLE works correctly in almost all modes. The one exception is that when DRIBBLE is in the append mode, the DELETE key does not work (back up and delete) as it should.

5.2. OPENFILE

Interlisp-VAX successfully handles the problem posed by the UNIX operating system of not allowing version numbers in file names (see page 15). However, Interlisp's method of handling the situation introduces another slight problem.

Because of the importance of the character "/" in specifying directory paths, the UNIX system does not allow that character in file or directory names. The ASCII representation for "/" is the number 47. If an OPENFILE is performed on a file with version number 47, the current UNIX operating system perceives it as the illegal character "/" and will not open the file.

5.3. PRINTNUM

Interlisp-VAX's PRINTNUM function merely prints the number without doing any rounding. It will be corrected in future versions.

6. Fixed Bugs

The following is a partial list of bugs that have been fixed:

6.1. BOUNDP

BOUNDP of something that is not a LITATOM returns NIL instead of blowing up.

6.2. COPYBYTES

COPYBYTES now works when the output is the terminal.

6.3. DWIMIFY

(SET 'FOO 1 + 2) will dwimify.

6.4. PEEKC

PEECK and READC return a carriage return when the input is a line feed and the device is the terminal.

6.5. UNPACK

UNPACK of a number now works.

6.6. MISCELLANEOUS

Also various CAR errors have been fixed.

7. Upcoming Attractions

Below is a summary of a few Lispusers packages, functions, etc., which Interlisp-VAX does not currently support but which will be included in future versions.

7.1. Hash Package

The HASH package, along with the WHEREIS package that depends on it, are still under development.

7.2. SUBSYS

A SUBSYS for Interlisp-VAX is currently under development and will be available at a later date.

7.3. PMAP

There is a PMAP package in Interlisp-VAX, but it is not yet documented.

7.4. EXEC

Interlisp-VAX's EXEC package, to be included in future editions, will employ standard UNIX shell commands instead of the TENEX or TOPS-20 commands described in the Interlisp-10 manual.

7.5. Improved Code

The Interlisp-VAX compiler is presently undergoing changes to produce better, much more optimized code and open code in future releases.

7.6. File Names

Should the UNIX operating system ever be changed to accommodate longer file names, Interlisp-VAX will be modified to handle them.

7.7. File Versions

Currently, Interlisp-VAX allows version numbers only up to 127; later releases will handle larger numbers. Also, in this version of Interlisp-VAX, the integer to which the environment variable VERSIONS is set merely enables the modified ls to show the version numbers. In future editions this integer will also denote the number of versions to keep for each file in the user's directory.

References

1. Bates, R. L., Dyer, D. and Koomen, J. A. G. M., "Implementation of Interlisp on the VAX," in *Conference Record of the 1982 ACM Symposium on LISP and Functional Programming*, pp. 81-87, Pittsburgh, Pennsylvania, August 1982.

2. Kashtan, D. L., Installing EUNICE, Version 2.0 (DRAFT). SRI International

3. Masinter, L. and Deutsch, L. P., "Local optimization in a complier for stack-based LISP machines," in *Proceedings of the 1980 LISP Conference*, pp. 223-230, Stanford, Calif., August 1980.

4. Moore, J S., *The Interlisp Virtual Machine Specification*, Xerox Palo Alto Research Center Report, Technical Report CSL 76-5, March 1979.

5. Teitelman, W., *Interlisp Reference Manual*, Palo Alto, California, 1978.

Index